B-007

変更管理のためのコード等価性検査方式の提案

An Approach of Equivalence Checking Method for Change Management

王　古玥†　　磯田　誠†　　外山　正勝†
Guyue Wang　　Makoto Isoda　　Masakatsu Toyama

1. はじめに

現在の組込み S/W 開発では，仕様変更やリファクタリングなどによってソースコードを変更することが多発する．変更前後ソースコードのテキスト差分から変更の影響範囲を正確に特定するのが困難なため，必要以上に広い範囲のテストを実施しており，作業工数が大幅に増加するという課題がある．本稿では，上記の課題に対する解決策として，変更管理プロセスで求められる影響分析を実現するために，テキストの見た目の修正ではなく処理内容を修正した箇所を判別するコード等価性検査方式を提案する．また，ツールの試作とソースコードへの適用を通じて発見した本方式の問題点と対策の検討結果を報告する．

2. 等価性検査を組み込んだ開発プロセス

本章では，標準的な開発プロセスの共通フレーム 2013[1]をベースに，コード等価性検査を組み込んだ開発プロセスを規定する．コード等価性検査とは，2 つのソースコードの処理内容が論理的に同じことを解析するための技術である（詳細は 3 章で示す）．

本方式の適用においては，ソフトウェア製作およびソフトウェア単体テストに焦点を当てる．開発現場に導入しやすいように実施内容を具体化し，課題となる部分にコード等価性検査方式を組み込む．

図 1 に示すように，仕様変更時はソフトウェア製作のフェーズの最初と最後に「影響範囲見積り」と「影響範囲特定」を実施し，ソフトウェア単体テストでは仕様に基づくテストを実施する．また，リファクタリングでは「影響範囲見積り」と「影響範囲特定」の実施は不要で，ソフトウェア単体テストでは回帰テストを実施する．コード等価性検査を組み込んだ開発プロセスの実施内容を以下に示す．

- ソフトウェア製作

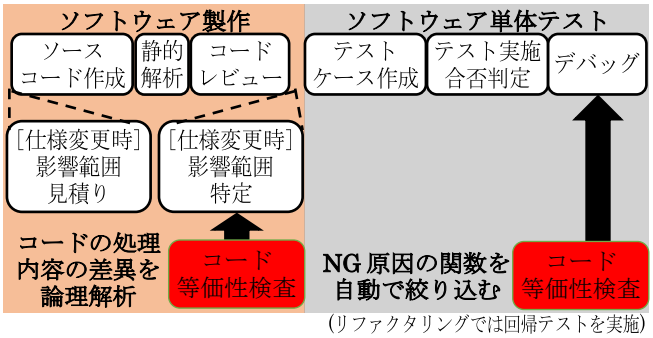
「影響範囲特定」でコード等価性検査を用いて，変更前後のソースコードの処理内容が論理的に同じことを解析する．従来のテキスト差分を用いる方法では処理内容と見た目の修正の判別が困難だったが，本手法ではこれらを明確に判別する．

- ソフトウェア単体テスト

「デバッグ」でコード等価性検査を用いて，テストが NG になった場合に原因となる関数を自動で絞り込む．検査結果が期待と異なった関数を原因の候補とする．従来は見た目を修正しただけでテスト NG の原因になる可能性が低い関数も調査対象にせざるを得なかったが，本手法では可能性が高い関数だけを調査対象にする．

このように等価性検査を組み込むことで，上流工程で欠陥流出を防止するとともに，下流工程でデバッグ時間を短縮する効果が得られると見込んでいる．

† 三菱電機株式会社　情報技術総合研究所　Information Technology R&D Center, Mitsubishi Electric Corporation



（リファクタリングでは回帰テストを実施）

図 1　コード等価性検査を組み込んだ開発プロセス

3. コード等価性検査方式

本章では，2 章で規定した開発プロセスのソフトウェア製作フェーズで用いるコード等価性検査の方式として，要求事項と実現方法，実現性を検討するための試作版ツールの特徴について述べる．

3.1 要求事項

コード等価性検査[2]は，仕様変更時にソースコードの処理内容が論理的に同じことを解析する．変更前後の 2 つのソースコードの間で対応付く関数を抽出し，新規作成または削除した関数は対応なしとした上で，図 2 に示すように判定する．

- 等価

2 つの関数に同じ入力値を与えたときに，常に同じ出力値が得られる場合，処理内容が論理的に同じである．

- 不等価

2 つの関数に同じ入力値を与えたときに，異なる出力値が得られることがある場合，処理内容が論理的に異なる．

関数が状態を持つ場合は，変更前後の 2 つの関数に同じ入力値列を与えたときに得られる出力値列を確認することで，状態を持たない場合と同様に等価であることを判定する．ここで，入力値列は時間軸を離散化したときのひと続きの入力値の集合，出力値列は入力値列に含まれる各入力値に対応する出力値の集合とする．

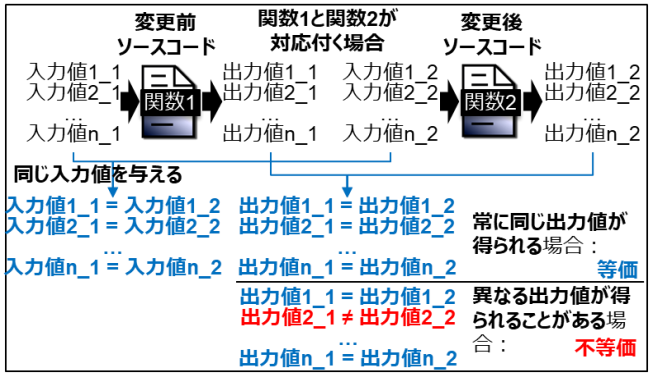


図 2　コード等価性検査の模式的な説明

3.2 実現方法

3.1 節の要求事項を実現するためのコード等価性検査手順を以下に示す.

(1) 入出力変数抽出

変更前後の 2 つのソースコードに含まれる関数ごとに，入力／出力変数を抽出する.

(2) 関数対応付け

変更前後の 2 つのソースコードの間で，入力／出力変数に基づいて対応付く関数を抽出する．新規作成または削除した関数は対応なしとする.

(3) 等価判定

2 つの対応付く関数が等価であることを判定する.

3.3 アルゴリズム

3.2 節に示したコード等価性検査手順を実行可能にするためのアルゴリズムを述べる．状態を持つ場合のアルゴリズムは持たない場合のアルゴリズムの一部のステップを修正して構成するため，先に状態を持たない場合のアルゴリズムを述べ，次に状態を持つ場合の差分を述べる.

➢ 関数が状態を持たない場合（図 3 を参照）

(1) 入出力変数抽出

ある変数から値を読み込んでいたら入力変数，ある変数に値を書き込んでいたら出力変数として抽出する.

入力変数の例：関数の引数や代入演算子「＝」の右側にある大域変数

出力変数の例：関数の戻り値や代入演算子「＝」の左側にある大域変数

(2) 関数対応付け

変更前のソースコードに含まれる関数と，変更後のソースコードに含まれる関数のすべての組合せについて，入力変数の名前と型および出力変数の名前と型が完全に一致する関数を抽出する.

変更後のソースコードで新規作成または削除した関数は対応なしとする.

(3) 等価判定

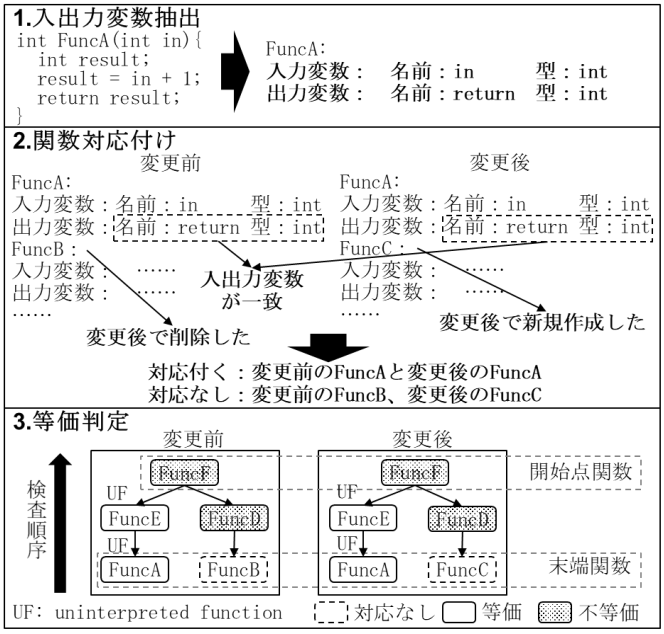


図 3 コード等価性検査のアルゴリズムの概要

2 つの対応付く関数の中で，開始点関数をルートとする関数コールグラフを対象に，呼び出し先がない末端の関数から呼び出し元に向かって，順々に等価であることを判定する．等価と判定した関数を呼び出している箇所は未解釈関数 (uninterpreted function)に置き換える.

➢ 関数が状態を持つ場合

変更前後の 2 つの関数が持つ状態が，ともに初期状態から始まって再び初期状態に戻るまで，すべてのステップについて等価判定する．ステップ(1)(2)は関数が状態を持たない場合と同様.

(3) 等価判定

開始点関数が初期状態に戻るまで複数ステップにわたって実行した場合の関数コールグラフを新たに作成する．この複数ステップ実行の関数コールグラフを対象に，関数が状態を持たない場合と同様に等価であることを判定する.

3.4 ソースコードへの適用

3.3 節の関数が状態を持たない場合のアルゴリズムを実現するツールを試作し，C 言語のサンプルコードと機器搭載の実物コードに適用した．サンプルコードは 26 個の数十行程度の独立したソースコードで構成する．その変更点は，変数や関数の名称変更，論理や計算結果が変わらない計算式や条件式，構文等の置換，配列アクセスをポインタアクセスの置換，処理の一部の関数化などを含む．実物コードは 1.5k 行程度と 3k 行程度の 2 個のソースコードで構成される．その変更点は，関数戻り値の型の変更，マクロの変更，引数の削除，大域変数の型の変更等を含む.

適用結果として，サンプルコードでは 3 個，実物コードはすべてにおいて，想定とは異なる判定結果となった．等価となるはずが不等価と判定されたケースとして，配列アクセスのポインタへの置換えがあった．その原因は，異なる入出力変数と認識されて対応なし関数になったことであった．また，逆に誤って等価と判定されたケースとして，ローカル変数の属性変更（static 化）があった．これは状態を持つ関数に対応できていなかったことが原因であった.

想定結果と違う原因を分析し，C 言語文法の問題，有界モデル検査ツール CBMC の使い方，試作版ツールの性能等の観点からコード等価性検査方式の問題点を 9 つ抽出して対策を検討した．5 件はツールの実装が不十分な部分を修正し，4 件は人手で情報を補うためのユーザインタフェースを追加する予定である.

4. おわりに

本稿では，コード等価性検査を組み込んだ開発プロセスを規定し，コード等価性検査方式の検討結果，およびソースコードへの適用を通じて発見した問題点と対策の検討結果を報告した.

今後は問題点に対処してコード等価性検査方式を改善し，実物コードへの適用範囲を拡大していく予定である．また，関数が状態を持つ場合も今後試作評価を実施する.

参考文献

[1] 共通フレーム 2013～経営者，業務部門とともに取組む「使える」システムの実現～, 独立行政法人情報処理推進機構（IPA），技術本部 ソフトウェア高信頼化センター（SEC）, 2013.

[2] R. Majumdar, et al. "Compositional equivalence checking for models and code of control systems," 52nd IEEE Conference on Decision and Control, pp. 1564-1571, 2013.